Virtual Tools: A Framework for Simplifying Sensory-Motor Control in Complex Robotic Systems

Randal C. Nelson, Martin Jägersand, and Olac Fuentes

Technical Report 576
March 1995

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

Virtual Tools: A Framework for Simplifying Sensory-Motor Control in Complex Robotic Systems

Randal C. Nelson, Martin Jägersand and Olac Fuentes
Department of Computer Science
University of Rochester
Rochester, New York 14627

March 15, 1995

Abstract

We describe and demonstrate a construct termed a "virtual tool", that provides a flexible interface to sensory-motor control. This interface is, from a user standpoint, substantially less complex, and more application-oriented than the raw devices. The basic idea is to use extra degrees of freedom present in a flexible system, in conjunction with sophisticated sensing (e.g. vision), to dynamically configure or "tailor" a manipulator so that it is matched to a particular situation and operation. This "virtual tool" is created by imposing customized, sensory modulated constraints between various degrees of freedom in the system. The remaining degrees of freedom constitute a small set of control parameters that are fitted to a particular operation. We argue that, within the confines of fairly broad application domains, a small set of "tool classes" can be defined that will serve as a general purpose sensory-motor toolbox for a wide variety of applications. We further argue that such class definitions can be made portable not only across tasks, but across platforms as well. The implementation of a number of basic tool classes, on various platforms, using vision and other sensory modalities, is described; and their use in performing multi-stage sensory-modulated manipulation tasks is illustrated. [1]

[1] This research was supported by ONR grant number N00014-93-I-0221.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	technical report

4. TITLE AND SUBTITLE

Virtual Tools: A Framework for Simplifying Sensory-Motor Control in Robotic Systems

5. FUNDING NUMBERS

ONR N00014-93-I-0221

6. AUTHOR(S)

R.C. Nelson, M. Jägersand, and O. Fuentes

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESSES

Computer Science Dept.
734 Computer Studies Bldg.
University of Rochester
Rochester NY 14627-0226

8. PERFORMING ORGANIZATION

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESSES(ES)

Office of Naval Research
Information Systems
Arlington VA 22217

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

TR 576

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Distribution of this document is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

(see title page)

14. SUBJECT TERMS

sensory-motor control; visual robotics; robot user interfaces

15. NUMBER OF PAGES

25 pages

16. PRICE CODE

free to sponsors; else $2.00

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	UL

1 Introduction

1.1 A Difficult Domain: Automation in Poorly Modeled Environments

In the last decade, robots of increasingly sophisticated mechanical design have been constructed. In domains where accurate geometric models of the world can be obtained and good kinematic models of the robot exist, a great deal of progress has been made in devising control schemes that allow such devices to be used in applications. Programmable manipulation devices are now routinely incorporated into industrial operations, from automobile manufacture to electronic component assembly, where they perform operations that were previously performed by customized, cam actuated devices, or in some cases, by human workers. The great advantage of such schemes in industry is that considerable retooling can be done in software rather than in hardware. The programming of these systems still tends to be a bit of arcane art, since the devices are quite complex; however, the underlying geometry imposes a rigorous framework, within which such programming can be carried out. In research labs, the state of the art has progressed considerably further. Mathematical techniques have been developed that permit complex devices with several non-linearly interacting degrees of freedom to be controlled in regimes where dynamic factors dominate, provided again, that accurate kinematic and environmental models are available.

In domains where accurate geometric models are not available, programmable mechanisms have had decidedly less impact. This is not due to lack of applications, though they lie less in production manufacturing than in other areas; for instance salvage, exploration, cleanup, and assembly/disassembly in environments that are hazardous, remote, or too large or small in scale for human operators to function in efficiently. Mining and excavation of hazardous waste sites have also been proposed as potential applications, as have devices to aid the handicapped, and even assist in surgery. The fact that tele-operated devices have been used to good effect in many of the above applications demonstrates that the mechanical technology needed for such operations already exists (though it could probably be improved). There is no lack of demand for usable technology in this area. Teleoperation is a difficult, error prone, and highly skilled occupation. Incorporation of some degree of automation, from semi-autonomous "tele-assisted" control, to full automation of significant tasks or subtasks, would substantially reduce the demands on the operator.

1.2 A Problem: Complexity

A fundamental problem in sensory-motor control is system complexity. The mechanical systems are complex; often having highly non-linear kinematics, problems with singularities, redundant degrees of freedom and multiple devices. The sensors, and the algorithms used to extract information from them are complex and often ill-defined, especially in the case of visual sensors; and there are many different sensory modalities. The interfaces to the mechanical devices are frequently arcane, generally close to the hardware, and not in a form that maps easily onto natural application descriptions. The formalisms for dealing with flexible mechanical devices in geometrically modeled domains are complex enough in themselves, but once they are mastered, a user can program a flexible mechanism for an application in such a domain. In order to apply sophisticated sensory-motor techniques however, a potential user must simultaneously apply expertise in robot kinematics, general control, and machine perception. This is asking a lot. Finally, there is the issue of task description. In a certain sense, it is perhaps the central issue; how to describe a complex sensory-motor task in a way that is convenient to the user, formally well defined, and robustly implementable across different devices and sensor configurations. If this issue can be satisfactorily addressed, the others may well fall into place.

The absence, in our domain, of prior models means that information about the environment (and in some cases, about the mechanism) must be acquired through sensory modalities. Such information comes in a number of forms, and only occasionally is it in the form of calibrated geometric attributes. For this reason, the techniques that have been developed for control in well specified environments cannot be applied directly. In the last decade, however, significant progress has been made in the development of sophisticated artificial sensing modalities, especially in the case of machine vision. Although vision is not by any means a solved problem, algorithms for tracking, stereo, motion analysis, object recognition, and others capabilities have exhibited good performance in a number of situations, especially in constrained environments. Coupling particular sensory abilities with robot hardware has produced several impressive laboratory demonstrations of sophisticated sensory-motor control. Despite these successes, such technology has not seen substantial use in practical applications. We believe that a primary reason for this is the lack of a unifying superstructure for hiding, from a user standpoint, the complexity inherent in sophisticated sensory-motor systems and packaging them into standard, usable chunks.

1.3 A (Partial) Solution: Structuring through Virtual Tools

The solution to complexity is organization. In this paper, we describe a construct termed a *virtual tool* that provides a means of reducing the complexity of designing, describing, directing, and analyzing the performance, of sophisticated sensory-motor systems. The construct is particularly useful in dealing with visual-motor control and systems with redundant degrees of freedom. The basic approach is simple: reduce complexity by hiding, at any given moment, whatever you don't need to know about the system, e.g., redundant mechanical freedoms, extraneous sensory information, awkward control bases etc. The reason for thinking that such an approach might be feasible stems from the observation that a lot of manipulation (and other) tasks have momentary descriptions that, with appropriate context, are very simple. Rotate the housing 90 degrees. Move lever up. Place gasket on manifold. Tighten bolt. Press firmly together. Align components. Compress fitting. Bend to shape. Advance three clicks. Slide into place. Fill to line. Pick it up. Insert it. Pull it out. Screw it on. Drive Forward. Climb up the mountain. The point is, the basic operations that people are accustomed to often use descriptions that involve only one or two objects, and movement or deformation in a single conceptual degree of freedom.

We would like to utilize a similarly simple description when dealing with artificial sensory-motor systems. Our basic idea is to use extra degrees of freedom available in a flexible mechanical system, in conjunction with sensory feedback, to dynamically configure the system so that, from user standpoint, it looks like a special-purpose device for a particular task. This *virtual tool* is created by imposing customized, sensory modulated constraints between various degrees of freedom in the system. The remaining degrees of freedom (if any) constitute a small set of control parameters that are fitted to a particular operation. The sensory modulation is crucial, because it permits virtual tools that cannot be defined in terms of fixed, low-dimensional loci in the manipulator configuration space. Virtual tools can be grouped into classes exhibiting common functionality, but distinguished by particulars of their configuration. The notion of a tool class is a fundamental structuring concept, and is defined more formally in the next section.

A simple mechanical example of the virtual tool idea is a crescent wrench, which represents a class of virtual wrenches. However, in a computer controlled manipulator, a configuration need not take the form of static settings of particular degrees of freedom, but can consist of more general relationships. We will show a number of examples of this in subsequent sections. Mathematically, of course, this is simply reducing the dimensionality of the system through constraints, which is a classic technique. What is new, is the idea of having a large set of predefined constraint classes

that are dynamically selected and imposed in response to the demands of the current task on the basis of available sensory information. One way of looking at the approach is that, instead of forcing the user to utilize a single mode of description (e.g. 3-D coordinates) in which all tasks can be described but none can be described easily, we provide a multitude of descriptions, (virtual tool classes) each of which make it easy to describe a particular sort of task. As an operation progresses through different phases, different descriptive modes will be successively brought into play. Such tools could be used in various modes, from elements of a programming language to semi-autonomous (tele-assisted) operations, to primitives used by an intelligent controller in more fully autonomous systems. Several such uses in will be illustrated in the following sections.

All this is not a lot of help if the tools must be re-implemented for each application. However, we argue that within the confines of fairly broad application domains (e.g assembly, custodial operations, outdoor navigation), a small set of "tool classes" can be defined, perhaps no more than a dozen or two, that will serve as a general purpose sensory-motor toolbox for a wide variety of applications. Moreover, we believe that these tool classes can be made broadly portable across platforms. Section 3.3 describes in detail why we believe this is possible, and we describe several examples later, where we have ported tool classes, with preservation of semantics, to radically different platforms. Basically however, the idea is that functionality is, to a large extent, device independent, and that functionally derived primitives should be implementable on any device satisfying specified minimum requirements. The overall organization we envision for using virtual tool classes in application domains is illustrated in Figure 1.

1.4 Overview

In order to be successful, such an approach must satisfy certain conditions. First, useful tools must be implementable using available technology. Second, the tools must be reusable across applications. In particular, there should be a relatively small "toolbox" of basic tool classes that will suffice to handle a substantial percentage of the situations which the user will encounter. (Obviously if every particular situation requires the development of a completely new tool, no efficiency is gained). Third, low-level details of the instantiation of tools in specific instances must be performed automatically. It is reasonable for a high-level user to specify that a certain grasp tool be used to pick up an object, but not to provide numerical information on the dimensions and position of the object. This is where the use of sensory information comes into play. Finally, the tool class definitions must be such that they can be made portable between different mechanical and sensory platforms. Moreover, it must be possible to provide a system of specification for quantifying the variations that will inevitably occur between implementations on different systems.

In this paper, we provide strong evidence in terms of implemented systems, for the first three points. Somewhat weaker evidence is presented for the fourth point, mainly because our access to different platforms is somewhat limited. The remainder of the paper is organized as follows. Section 2 briefly establishes a historical context, and surveys some supporting work. Section 3 formalizes the basic definitions for tools and tool classes, and discusses the use of visually modulated tools. Section 4 describes the definition and implementation of a number of tool classes for the domain of visually-controlled placement and assembly, illustrating the practicality of implementing useful tools using current technology. Section 5 shows these tools working in conjunction to perform different extended operations, thus illustrating portability between applications. Section 6 discusses portability between platforms.

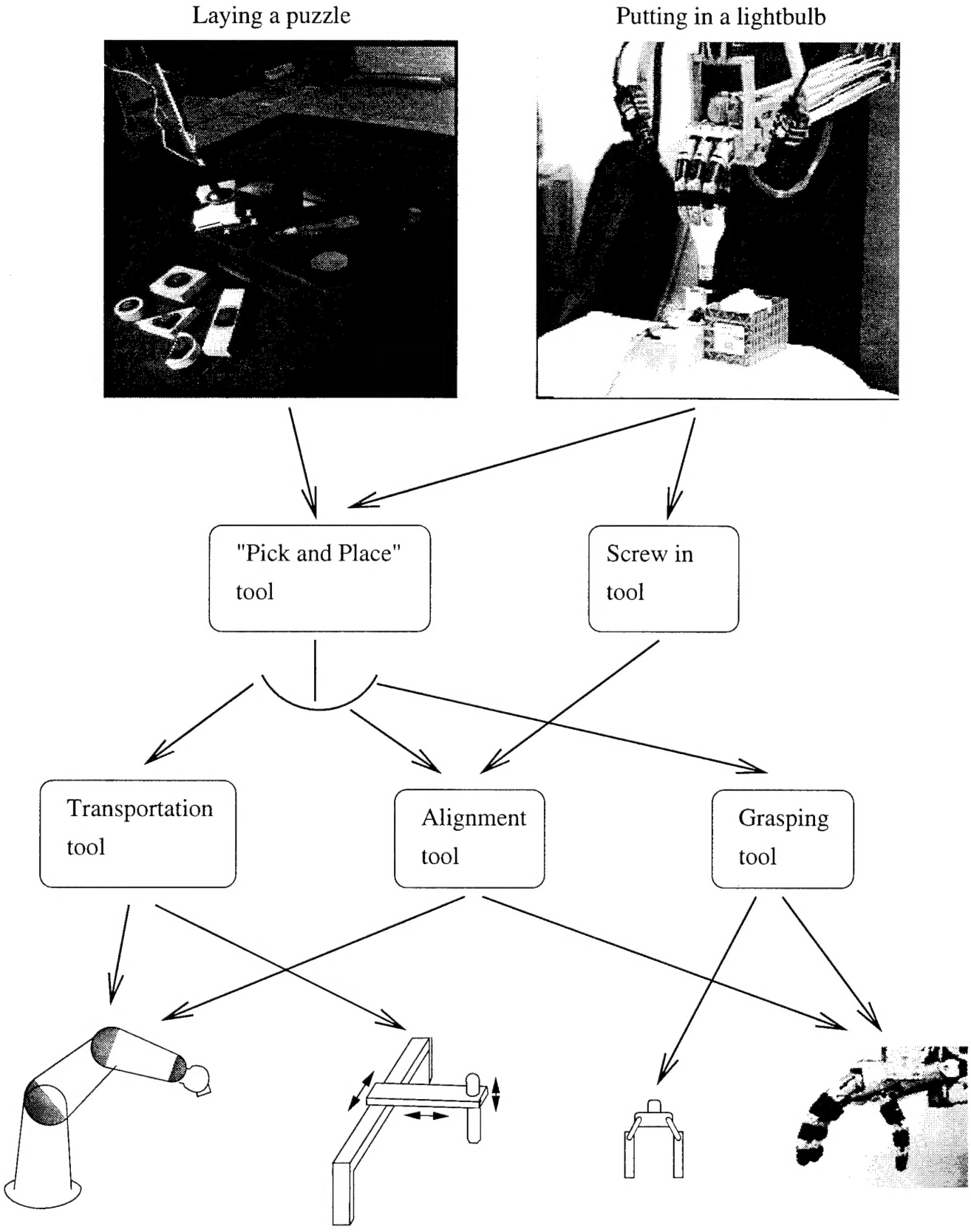


Figure 1: Implementation platforms and use of virtual tool classes

2 Historical Context

This paper draws on a large amount of previous work. The approach has it roots in work on active or behavioral vision; the connecting thread being the idea that perception is intimately tied up with action. Recent work on active vision [1; 8; 6; 7], and more generally, active perception [4; 5] has addressed how directed control of the sensor characteristics (e.g. eyes, or tactile receptors) can simplify the process of obtaining the desired information. Most work to date has focussed on the effect of the ability to move the sensor [12; 35] or dynamically change an internal focus of attention [30]. Work on purposive or behavioral vision [2; 27; 26; 22] attempts to take the context of the task explicitly into account. The work described here is an application of these ideas to the areas of manipulation and functional model acquisition.

Our method is also somewhat related to work on redundant manipulation, the connection being that redundancy resolution, like virtual tool instantiation, reduces complexity in a system by imposing internal constraints. Most of this work makes little explicit use of functional considerations, and has centered on finding efficient mathematical methods for regularizing the inverse kinematic problem [23; 37; 17; 37]. Several researchers have addressed the idea of using the extra degrees of freedom to accomplish several goals at once, which is functional in a somewhat different way [25; 37; 10; 24]. Task based specific reduction of the dimensionality of the space has appeared in couple of special cases, most notably in Raibert's notion of a virtual leg [29], and the idea of viewing a hand as several more specialized manipulators lumped together was discussed by Iberall [19] in the context of grasp classification. However, the idea of flexible, task dependent dimensionality reduction as described here does not appear to have been utilized as a general structuring mechanism.

In our implementation of specific visually-modulated tool classes, we make use of techniques for differential visual feedback control that were pioneered primarily by Weiss and Sanderson [31; 32; 33]. Since then, real world implementations have been made by a number of researchers, e.g., [11; 14; 13; 34; 16; 38]. An adaptive controller, which is key to making some of the visual implementations work well without prior models was developed recently here at Rochester [21] and in Japan [18]. Previous approaches either assumed that the system need only be calibrated once (e.g. [9]) using a set of specific "test movements", that it could be decoupled into a set of single variable adaptive controllers (e.g. [33]), or that the system could be modeled using an ARMAX model (e.g. [14]).

At the higher level end of task specification and trajectory planning, we have been inspired by the user friendly "learn-by-watching" interfaces found in [20; 36; 28]. These approaches are close to ours in spirit, in their efforts to provide usable interfaces to complex devices, but they address a higher level of task acquisition than the fundamental tools we describe here. However, these approaches could very well make use of virtual tool classes as a vocabulary to express what they observe. The lack of a effective vocabulary is one of the primary challenges in such systems.

3 Basic Definitions

3.1 Virtual Tools

The basic idea of a virtual tool is drawn from the observation that the most efficient way of performing a task is generally to use a device that is specially designed for it. This is a fundamental principle of hard automation, and a primary reason that highly flexible robots have not, so far, been in much demand for mass production operations. The reason for this is that most tasks, even very complicated ones, can be reduced to a sequence of operations that require the control of only a few, carefully engineered degrees of freedom. This suggests that a way to use extra degrees of freedom in a system is to use them to customize, or "tailor" the system so that it becomes, in essence, a

special purpose tool for the task at hand. We call the resulting instantiation a *virtual tool.* Formally, this tailoring takes the form of implementing constraints on the degrees of freedom of the system through lower-level control processes. Remaining degrees of freedom appear as control parameters of the virtual tool, and can be used by higher-level processes, or by a human teleoperator, to control the system. Since the control parameters are directly linked to the task at hand and are of low dimensionality, the control problem can be greatly simplified. Furthermore, the device can shift facilely from one virtual tool to another as the system progresses through the stages of a complex task.

As mentioned above, an actively controlled system, has the advantage that it is possible to impose constraints more general that simply fixing a mechanical degree of freedom as in an adjustable wrench. For example, two adjacent joints in a planar chain linkage could be constrained to maintain equal angles, and the two degrees of freedom replaced by a single one specifying the total deviation through the two joints. A more complicated example would be constraining two 6DOF manipulators to maintain a constant distance between manipulator endpoints—a virtual link. The addition of sensory information is the final generalization. For example, a manipulator could be constrained to move in a particular plane while remaining a fixed distance from some surface. There is a single controllable degree of freedom in this system; however, it cannot be predefined in terms of purely kinematic constraints.

The idea of a virtual tool is similar to, and takes its name from some techniques that have been described previously for dealing with some highly specific situations. An example is the virtual finger introduced by Arbib *et al.* [3], which is composed of one or more real fingers working together to perform a task. [19] describes how a hand can be used as three different grippers by changing the mapping of virtual fingers to real fingers. Similarly, the concept of virtual legs has been used, e.g. by Raibert [29] to derive two and four legged gaits from algorithms originally designed for a one-legged hoping robot. The idea of using the reduction as a general method for packaging sensory-motor devices, however, was not developed in either case.

3.2 Virtual Tool Classes

As described above, a single virtual tool is not very useful as a general device, since it only applies to a specific situation. The tools used for rotating two different sized objects using the fingers of a robot hand, for example, will most likely involve slightly different constraints. What is needed, is an effective means of instantiating the constraints defining appropriate virtual tools. Since a user cannot be expected to provide the necessary constraints at a high level of detail, the process must be largely automatic, and definable by a few parameters that can be easily supplied by the tool user (man or program). In many cases the information needed for instantiation will be derived from sensor data.

Instantiation involves two natural levels of description. At the top level is a description that corresponds to a class of tools that are used in similar ways. At the lower level is a set of discrete parameters that describe how the particular tool is fitted to a particular instance of the problem. For example, in the domain of hand tools, a top level description might specify use of a 12 point box wrench (as opposed to a hammer, a punch, or an allen wrench). The lower level description would specify that it is a 5mm wrench. Both these decisions can be made on the basis of sensory information, but rather different levels of processing and decision-making are involved.

This dichotomy provides the basis for a fundamental structuring element: the *virtual tool class.* Intuitively, tool classes correspond to natural functional classes described above. Formally, we can define a virtual tool class as a set of virtual tools indexed by a set of *instantiation parameters* which modify the defining constraints in a predefined manner. Usually, the instantiation parameters will

consist of a small number of variables taking on continuous values in a fixed range, for example, the width of a wrench tool, or finger setpoints in a grasp tool, but there are a few examples where some other enumeration scheme may be appropriate. The value of the tool class as a structuring element arises from the fact that between-class distinctions correspond to functional distinctions in the world, thus making tool classes a natural primitive for a task-oriented language. The instantiation parameters, together with any tailored control parameters form the user interface for a tool class.

The instantiation of tools within a class is conceptually straightforward to automate. The basic approach is to include instantiation procedures as part of the class definition. Specifically, class definition provides a-priori, a mathematical form that the constraints will take, along with a number of free *instantiation parameters* defining them. For example, consider a tool to execute a grasp with a robot hand. In the simplest case this tool might have single associated control parameter that would essentially represent the degree of closure. Position, orientation, and initial closure would represent instantiation parameters to be initialized on the basis of sensory measurements of the object to be grasped. In this example, instantiation places the tool in the correct orientation with respect to the the object to be grasped. The closure parameter could then be decreased by the system until sensory input (e.g. tactile sensors) indicated that the object was securely grasped. Or the closure control parameter could be eliminated entirely, and replaced by instantiation parameters specifying force of grasp and speed of closure.

3.3 Class Structure

We believe that virtual tool classes can provide a structuring element that is transportable across both applications and environments. The idea is feasible because, although a large number of different manipulators and sensing devices are available, the raw control and sensing interfaces are dominated by a few modalities. On the control side there is position, velocity, and force control. On the sensory side there are position, velocity, force, and imaging sensors. This raises the likelihood that a set of generally useful tool classes can be defined in a way that is, to some extent, device independent. Not every tool class could be implemented in every mechanism, of course, but the basic manipulator types (grasp devices, transport devices, local manipulators) are few enough so as to allow broad generalizability.

For any specific device of course, some re-implementation of the tools will be necessary; the ideas is that this would be like providing a compiler for a new machine architecture (only hopefully easier). Old implementations could be substantially reused, and given the existence of the appropriate control and sensory modalities, implementability could be guaranteed for any device with sufficient degrees of freedom and adequate workspace. There would also be some variation in the limiting capabilities of the same tool on different devices (due to variation in size, speed, power, etc). The idea is that most of this variation could be covered with a class specific set of specifications, that would have different values on different platforms.

The following template is an outline of the structure we have used in defining virtual tool classes. This template is not intended as an artifact for formal proofs, but rather as instructions to the human implementer and user. In particular, descriptive categories such as preconditions and unconstrained degrees of freedom are not intended to be inclusive sets (doing so would drag in the frame problem), but as pointers to especially critical factors.

Template for tool class definition

- Resource requirements: Constraints on hardware and software components needed to implement a tool class.

1. Mechanical hardware: Constraints on the mechanism. e.g., how many degrees of spatial freedom are required? are oppositional freedoms needed?
2. Mechanical software: Usually low-level control software. Is force control required? velocity?
3. Sensory hardware: Sensory devices required by the tool. Force, position, imaging sensors, and minimum resolution if applicable.
4. Sensory Software: Sensory processes required, e.g. visual tracking, recognition abilities, force profile analysis.
5. Other Tools: Tool classes may be defined hierarchically. Though technically, other virtual tools are composed of mechanical and sensory resources, we want to distinguish cases where these resources have already been organized at some level.

- Resource Specifications: These are the numbers that distinguish the performance between instantiations of the same tool on different systems.

1. Mechanical hardware specs: Workspace, payload etc.
2. Mechanical software specs: Technical limitations on control system, update rate, etc.
3. Sensory hardware specs: Camera resolution, tactile sensitivity, etc.
4. Sensory software specs: Recognition error rates in domain, tracker update rate and maximum velocity, noise in pressure sensors, etc.

- Functional Description: High level description of functionality

1. Nominal effect: Detailed description of tool function. What is done, grasp, move, align, etc., with what constraints and under what form of guidance.
2. Notable unconstrained parameters: In some tool definitions certain functionally irrelevant, but easily controllable freedoms may be left unconstrained. For example, a transport tool might explicitly state that the changes in orientation of the transport class are not constrained by the definition.
3. Success preconditions: Significant conditions that must hold in order for nominal effect to take place. These are necessary, not sufficient conditions, intended to inform the user about critical factors that are *likely* to affect tool performance.
4. Functional specs: Accuracy specifications that can be associated with tool use.
5. Failure modes: What happens under certain modes of failure (e.g. loss of visual tracking. grasped object dropped, impossible commands). Again, not intended to be inclusive.

- Functional interface: This is how the user, whether human or computer process, actually make use of a tool.

1. Instantiation preconditions: Conditions that must hold before a tool of this class can be instantiated. For example, a robot hand must grasp an object before it can manipulate it. These are usage hints, not an inclusive list.
2. Instantiation parameters: The numbers (or flags) that specify a particular tool in a tool class. The instantiation procedure uses these parameters to initiate the low-level processes that define the virtual tool.

3. Control parameters: These are tailored degrees of freedom that are attached to some tools. Many tools may have no control parameters. Such tools can be treated as standalone processes that perform some task and then return. A tool with control parameters, on the other hand, must typically be embedded in some sort of control system.
4. Output parameters: User readable parameters, representing sensory data or some aspect of internal tool state. Strictly speaking, they are unnecessary, as a user may run an arbitrary sensory process in conjunction with a tool without affecting its operation. However, for some tool classes, there may be state or sensory attributes that are frequently needed when tools from the class are used (e.g. torque on a wrench tool).

3.4 Toolboxes and Tool Transformations

Generally, a sophisticated operation will involve the use of more than one virtual tool. We believe that for rather broad domains, there exists a relatively small set of virtual tool classes, perhaps no more than a few dozen, that will cover a significant proportion of the situations encountered in any particular operation in that domain. Of course, there will often be cases where custom tools are required, but our position is that having the right virtual toolbox can eliminate a very large proportion of the implementation effort.

As an operation progresses, a sensory motor-system will frequently need to switch from one virtual tool to another. From a user standpoint, this transition must take place seamlessly. This is accomplished through a *tool transformation*, which amounts to a highly constrained change of control basis. In particular, since virtual tools are associated with control parameters that change the physical configuration of the robot device, it is possible for two different virtual tools to have the same physical configuration for certain settings of the control parameters. At this point, a seamless transition can be made, and a different set of constraints swapped in. There is no physical discontinuity, but we now have a different set of control parameters, and different constraints.

Such a tool transformation instantiates a new tool at the transition point, which requires specification of a set of instantiation parameters. Some of these parameters can be determined from the condition requiring physical continuity at the transition. The others must be extracted from sensory data. Specific procedures on how transformations are accomplished must thus be provided at the level of the tool box. In particular, when a virtual tool class is incorporated into a toolbox, information must be provided regarding what transformations are possible, and what constraints hold between instantiation parameters in each case. Section 6 provides some examples of this operation, but assembly of a fully integrated toolbox is a topic of ongoing research.

3.5 Sensory Integration

In the preceding material, we have discussed the incorporation of sensory information into the definition of a virtual tool, but have not addressed how this might be accomplished. To do this we propose to utilize a perceptually based model of sensory control, which is described below.

Traditional approaches to robotics control have utilized explicit geometric models of the world. These models serve as a central "clearing house" between perception and action. Perceptual processes update the model, goals are specified in terms of desired states of the model, and actions are described in terms of transformations of the model. The problem with this approach has been that models have been chosen on the basis of their mathematical or transformational simplicity, with (at most) secondary consideration given as to whether real-world perceptual and motor processes can be easily mapped to the central model. The result has been "failure of perception or modeling" in

which actual perceptual processes and motor devices fail to conform to the abstract specifications of the model, resulting in poor performance of the system.

The virtual tool framework allows us to employ perceptual information more directly. The basic idea is to describe the perceptual constraints associated with a tool class in terms of sensory parameters that are actually available rather than in terms of an abstract spatial model that we hope can be derived from sensory data. For example, a tool that involves visually verified placement can utilize information specified in image space rather than in a separate geometrical space which must be accurately calibrated in order to work. The elimination of the requirement of a single central representational model allows us to stay much closer to the information available from the sensor. This does not imply that no higher-level perceptual processing need be performed. On the contrary, sophisticated processing will often be employed, but such processing will be in a constrained environment associated with a particular tool class, rather than being required to be generally valid. This is an enormous advantage, especially when using visual perception, since current technology has produced visual processes that work quite well in constrained domains, but has been considerably less successful in producing techniques that work in arbitrary environments.

Under this view, constraints associated with a virtual tool class may involve particular perceptual processes that are specified in the class definition. Furthermore, goals associated with the use of a tool can be specified in terms of the same perceptual processes. This leads to a model of task specification where actions are specified in terms of shifting sets of perceptual processes, rather than in terms of a single central model. In keeping with the parallel idea of "active perception", we term this approach *perceptual action.* The basic functional unit in this model is a "precondition perception, action, goal perception " primitive, where preconditions for action and process goals are both specified in terms of available perceptual primitives (e.g. in image space). This model is more fully discussed in [21].

Perceptual processes associated with a tool class serve several distinct functions. First, as mentioned above, they provide information that may be necessary for instantiating a specific instance of a tool class. Second, they provide the perceptual triggers that initiate the transformation from one tool to another during the execution of a complex operation. Third, they provide feedback information that is used to maintain the constraints that define a virtual tool. Finally, they provide a well conditioned interface through which manipulation with a single tool can be controlled by sensory-motor feedback.

3.6 Vision-based Tools

In our work, we have particularly emphasized visual sensing. Vision alone is not sufficient to implement a general toolbox, but it is a rich and flexible modality that has the potential to facilitate a wide range of operations. A tremendous amount of research has been done in the area of visual perception, and a large number of algorithms exist for different applications which, although not fully general, can be demonstrated to perform well under constrained conditions. This is exactly the situation we have in the application of a virtual tool: the perceptual processes are only required to work in a particular context. Visual processes that are potentially useful for defining, instantiating, or guiding virtual tools include feature detection, object recognition, shape analysis, motion analysis, and visual tracking.

A particularly promising area with respect to virtual tools is a general technique of uncalibrated visual control known as visual servoing. This has been around, in simple forms, for some time, but recent developments, at Rochester and elsewhere have demonstrated that the technique can be used in situations involving many degrees of freedom, with sophisticated visual features, and without any prior knowledge of camera parameters, world geometry, or robot kinematics.

The idea is to relate changes in scene or object appearance to a set of manipulator freedoms by means of a generalized Jacobian. In particular, suppose that the relevant factors concerning the appearance of a scene or object can be represented by a vector $\mathbf{x}$ of scalar quantities. These quantities might be the image coordinates of certain image features such as corners, or they might be the lengths, orientations, and centers of line segments, or parameters describing extracted blobs, or even colors. In general, any task-relevant scalar that is expected to change in a continuous way when the object is manipulated could be used as a feature. We also have a vector $\mathbf{y}$ representing the current controllable freedoms of the mechanical system. These might be standard manipulations like rotations about various axes, translations in various directions, or combinations of these. But they could also correspond to various deformations of the object; for instance bending or twisting, or articulation about a joint, or any of the tailored control parameters associated with a virtual tool.

To make the discussion concrete, suppose we are manipulating an object. For a particular pose of the object, we can describe the change in appearance of the object under a small manipulation by the partial derivatives of the feature values with respect to that manipulation. If we stack together the partial derivatives of all the feature values with respect to the various control freedoms we know about, we obtain a generalized Jacobian matrix $\mathbf{J}$ that relates changes in the appearance of the object to a general (small) manipulation composed out of the controllable freedoms by $\Delta\mathbf{x} = \mathbf{J}\Delta\mathbf{y}$. We refer to this as the *motor-visual Jacobian.* The Jacobian $\mathbf{J}$ has a nice property; it can be determined experimentally on line. In the most explicit case, the robot can pick up an object, look at it, make small manipulations along the controllable freedoms, and observe the changes in appearance. The terms of the Jacobian are approximated by ratios of differences.

However, the Jacobian can be determined more elegantly using an adaptive procedure starting from a guess. The procedure often works even for a very bad guess (i.e. a random one). The idea is to watch the actual effect of a commanded movement and compare it to the change that is predicted on the basis of the current Jacobian estimate. The difference can be used to correct inaccuracies projected along the direction of motion in appearance space. If the path followed has sufficient structure, the entire Jacobian can be kept up to date. In fact, it can be shown, that even if the path does not contain sufficient structure to determine the entire Jacobian, the approximation that is obtained has bad effects only out of the subspace spanned by the path, which means that the adaptive approach can be used to acquire and maintain a partial model that is precisely fitted to a particular manipulation. A more detailed description of the visual servoing algorithms we employ can be found in [21].

Adaptive visual servoing can be applied in conjunction with virtual tools in two ways. The first is externally, to visually control a tool through its control parameters (the tailored degrees of freedom that are associated with it). This is the traditional use of visual servoing, with the bonus that the control parameters are likely to be well conditioned due to the efforts of the designer of the tool class. The basic idea is to determine a Jacobian associated with the control parameters of the virtual tool that is currently instantiated. This could be done either directly, or adaptively starting with an approximation derived from prior knowledge about the virtual tool. We then somehow obtain a goal condition that describes the appearance of the object when the desired manipulation has been completed. The goal description could come from prior knowledge, from a high-level controller, or it could be constructed from high-level hints provided by an operator in a tele-assistance system. We then compute an appearance difference vector $\Delta\mathbf{x}$, solve for a manipulation vector $\Delta\mathbf{y}$ using the pseudoinverse of $\mathbf{J}$, and servo on this result using whatever control law we desire.

The second, and more interesting application, is to use visual servoing processes as part of the tool definition in constructing higher-level tools. For example, we might have a transport tool class,

where the control parameters include the desired object location in image space in a pair of live images, and whose effect is to move a grasped object along a straight line from its current position to the specified one. (Note that this is not a complete class description; we have not specified how, if at all, rotations are constrained, or anything about temporal behavior). Instianting an instance of this class then involves setting up the visual servoing machinery to move the object from the initial position to the goal, including starting appropriate tracking systems, and providing an initial guess for the Jacobian. During use of the tool, i.e. while the transport operation is executing, the visual servoing processes act to maintain the invariants associated with the tool - in this case the spatio-temporal constraints defining the desired path. It is this sort of use of visual processes that we are most interested in, because here they serve to hide complexity and make a system more usable.

4 Implemented Tool Classes

In order to demonstrate that our approach is practical using available hardware and sensory alorithms, we have implemented a number of virtual tool classes on hardware that we have available. On the robot side we have used a couple of PUMA robot arms with different link lengths, and a Utah/MIT dextrous hand. Vision sensing was provided by a a number of different CCD cameras. The hand comes with joint position and tendon tension sensors, and we have added tactile sensors to the fingertips, which are used in some of the grasp tools.

4.1 Grasp Tools

Our first example is a dextrous grasp tool class, intended for use with multi-fingered devices that have degrees of freedom remaining after the grasp is completed, allowing the grasped object to be manipulated. The tool definition does not refer to these additional degrees, and the tool class could be implemented on a parallel jaw gripper, but there it would be just another name for a simple close-the-jaws grip.

The mechanical requirements for the class are a device with two or more "fingers" and sufficient degrees of freedom so that, for any configuration where the freedoms are not at their limits, an opposition can be implemented between any pair of fingers. The Utah/MIT hand satisfies this requirement. So does a parallel jaw gripper. Hardware and low-level software compatibility with position and force control are also required. The sensory requirements are tactile contact sensors on the fingers that can specify whether or not the oppostion surface of the finger is in contact with an object, and some measurement of the normal forces at a fingertip.

The use of this tool (preconditions) assumes that the gripper is initially placed in a position where it can grasp the object. The instantiation parameters provided by the user (process or person) specify a gripper preshape (initial finger positions) appropriate for the object to be grasped, a minimum opposition force, and a closure speed. The class has no free control parameters. The operation of the tool is to close the fingers from the preshape positions towards a grasp center, until they contact the object, and then increase the forces on the fingers until the minimum opposition force is obtained in at least one finger. The particular implementation will guarantee that the maximum force at this point does not exceed some specification constant times the minimum force.

We implemented this grasp class on the Utah/MIT hand, using the analogue PD controllers in the hand for both position and force control, and Interlink sensors for tactile information. For specific applications, the above grasp tool class can be rolled inside of a more specialized class; for example, one where instead of requiring a preshape position for each finger, a single preshape parameter specifying degree of openness of the whole hand could be used. This specialized tool

can be used in cases where the objects are such that careful finger placement is not required for a secure grasp. For all the applications we have performed so far, the specialized version has proven adequate.

4.2 Rigid Manipulation Tools

The next example, is an instance of the use of a virtual tool class purely for the purpose of dealing with a highly redundant, and for the purposes of manipulation, poorly conditioned control basis. The Utah/MIT hand is a flexible manipulator having four fingers, each with four joints. The native control system consists of 16 independent PD controllers, one for each joint, implemented in analog circuitry. In order to make the hand more useful for object manipulation, we implemented a tool class for local, 6 DOF rigid manipulations of arbitrary grasped objects. This is a low-level tool, the equivalent of which comes packaged with most commercially available manipulators. Implementing this class on a Puma robot arm is trivial. Putting it on a flexible hand, however is more challenging. This is a simple example of a tool class that is portable across platforms. The fact that the achievable rotations and translations using the hand are quite limited compared to a PUMA is reflected in the specifications that accompany the particular implementation.

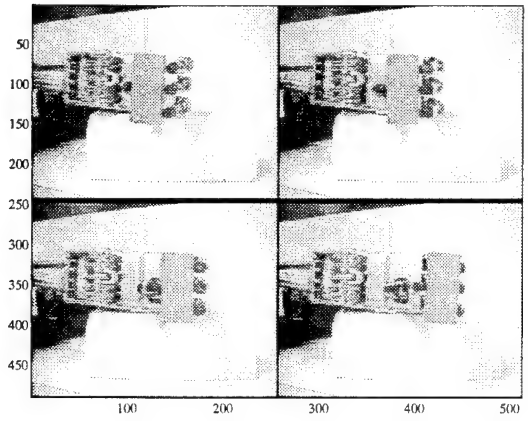
The system we implemented first uses a grasp tool, such as the one described above, to position the four fingers so that they securely grasp the particular object (preconditions). Haptic information about the position of the fingers then provides information allowing the instantiation of a virtual tool having six control parameters. These correspond to the three principal translations and the three independent components of a rotation matrix in the local coordinate system of the hand. This tool is adapted to the particular object grasped. More specifically, the instantiation parameters correspond to PD setpoint of the fingertips, and the invariants maintained correspond to rigid transformations of the tetrahedron defined by the setpoints. Use of the system is illustrated in Figure 2. Details are given in [15].

4.3 Visual Transport Tools

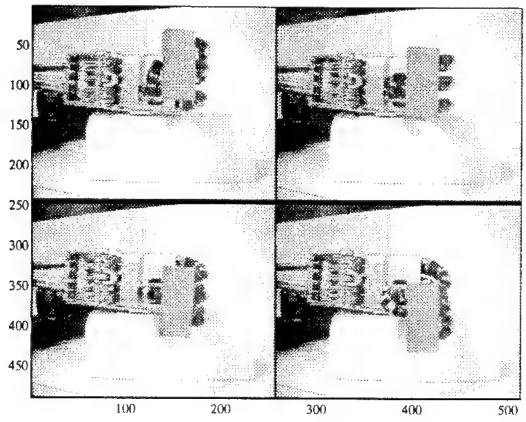
This example is a version of the transport tool class described previously. Essentially, it permits an object transportation movement to be specified in binocular image space. This tool would be of use to a tele-operator who could point to image locations, or as part of a more autonomous system, where image locations are provided from other perceptual processes, for example recognition or shape analysis algorithms. To give a taste of how it is done, the class definition in terms of the template described earlier is given below.

- Resource requirements:
    1. Mechanical hardware: Manipulator with at least three mechanical degrees of freedom spanning a non-zero 3-D volume.
    2. Mechanical software: Low level control software that can be made to support position and velocity control.
    3. Sensory hardware: Two live camera systems whose fields of view span the workspace. In this version cameras are assumed to be fixed for the duration of a particular tool instantiation.
    4. Sensory Software: Real-time, binocular visual point feature tracking.
    5. Other Tools: None.

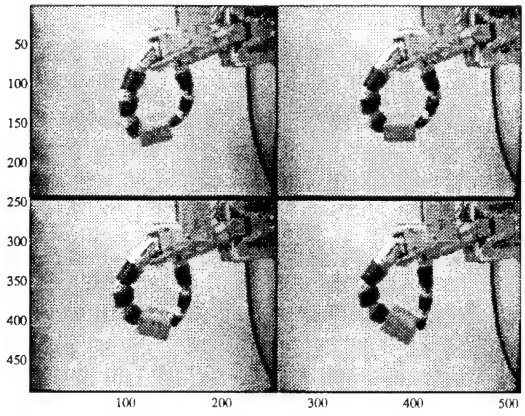
Translation along the X axis



Translation along the Y axis



Rotation about the Y axis



Rotation about the Z axis

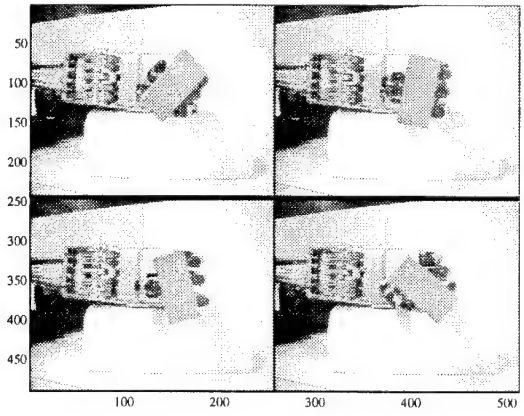


Figure 2: Examples of local manipulation performed by a rigid manipulation tool implemented on the Utah/MIT hand.

- Resource Specifications

  1. Mechanical hardware specs: Geometry of workspace and Manipulator specs (speed, payload, etc).
  2. Mechanical software specs: Temporal resolution of controller.
  3. Sensory hardware specs: resolution of cameras relative to workspace.
  4. Sensory software specs: Spatial and temporal resolution of tracking system.

- Functional Description

  1. Nominal effect: Move manipulator so that specified feature on object moves along lines in binocular image space to specified location.
  2. Unconstrained parameters: Orientation change of object. (Note however that implementations should try to make this small so that tracker does not fail due to rotational occlusion of tracked feature.)
  3. Success preconditions:
     - Line connecting points does not leave workspace.
     - Trackers do not lose tracked feature along line.
     - Visual goal must be physically possible.
     - Cameras should be placed so geometry is well conditioned.
     - No collisions.
  4. Functional specs:
     - Maximum deviation from linear trajectory.
     - Endpoint accuracy.
  5. Failure modes: System halts (we haven't got very sophisticated here yet).

- Functional interface

  1. Instantiation preconditions: Tracked feature is rigidly attached to manipulator
  2. Instantiation parameters:
     - Image positions in two cameras of feature to be tracked.
     - Corresponding image positions in two cameras of desired location of tracked feature.
     - Timing.
  3. Control parameters: None.
  4. Output parameters: None.

The definition does not specify how the class should be implemented. In particular, it does not specify that visual servoing be used to implement the visual-mechanical connection. If a particular installation has enough information available to allow model-based techniques to be used, then that is fine. However, the definition does guarantee a certain amount of flexibility, for instance, arbitrary camera placement (modulo ill-conditioned configurations) must be supported, and the user cannot be required to supply information about this. In our implementation of the tool class, we found that adaptive visual servoing was the most effective way to obtain the desired behavior.

Figure 3 shows an instantiation of this transport class in use. The point following a straight line in left and right images is the point that was tracked. The curved paths followed by the others are

a consequence of the lack of orientation constraint for this tool class. The implementation shown in the figure used a pair of cameras pointed at the manipulator. We also implemented the tool in a binocular eye-in-hand configuration, where the cameras are mounted on the manipulator. The usage is the same except that the manipulated object provides the un-moving point, and the goal location is tracked. This platform port was trivial. We used exactly the same code, interchanging only the indentification of the motionless and tracked features. Because of the adaptive nature of the visual servoing used in the original implementation, no changes were needed in order to acommodate the significant changes in geometry.

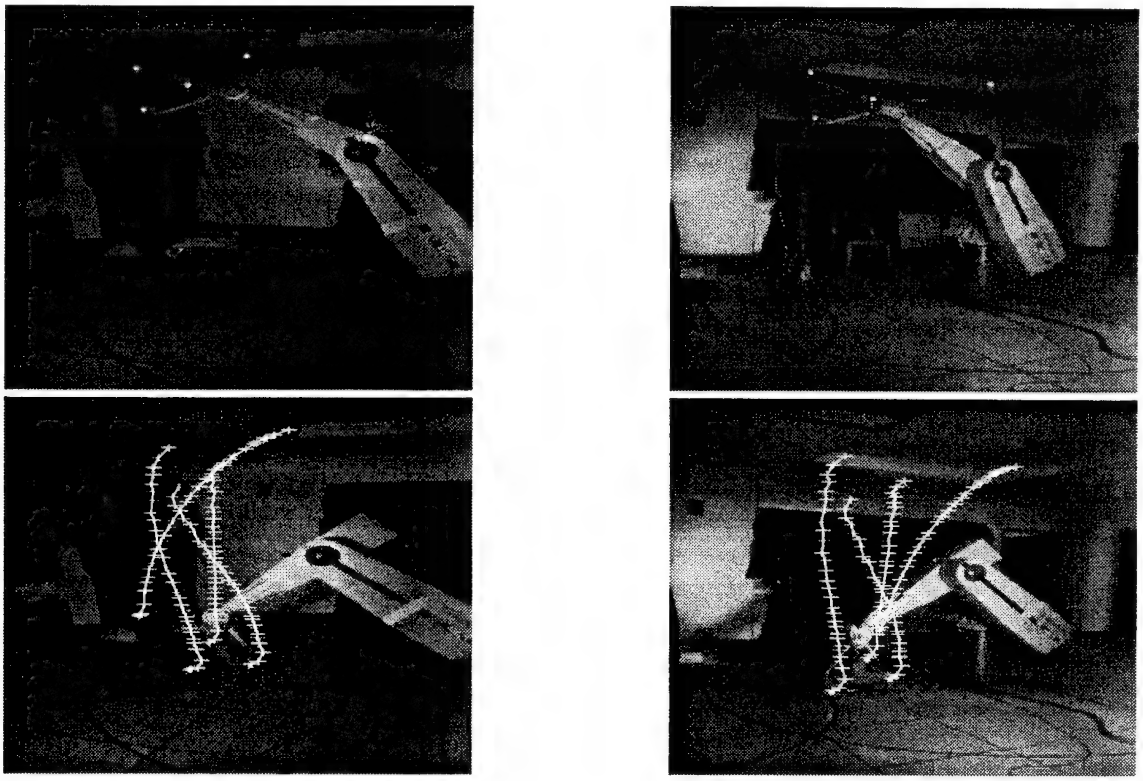


Figure 3: Use of a transport tool. Above: Initial configuration of robot as seen in the left and right cameras. Instantiation parameters specify a visual goal near the floor. Below: Goal configuration achieved by the controller. The image space trajectory of the tracked feature (straight line) and several others is overlaid.

4.4 Visual Alignment Tools

We also implemented a visual alignment tool. This is somewhat like the transport tool, only all six rigid degrees of freedom are constrained, which requires that at least three points be tracked in two images. Tracking more than three points gives better results, and a certain degree of robustness against tracker failure, so this class is defined to allow an arbitrary number of point features to be used in the instantiation of the tool. Also, unlike the transport tool, it is not necessary to provide correspondences between points in separate images. (The actual geometry is a bit complicated, and a minimal, well-conditioned solution can be obtained with e.g., three points in one image and one in a second; we choose to make it easy on the user and require at least three in each.)

Usage is similar to the transport tool. The user (man or machine) specifies current and goal

image location of a set of points, and the system does the rest. There is a potential difficulty associated with the fact that the specified goal must represent a possible state of the world. The intended use of this tool is in assembly applications, for insertion or mating applications, where the goal locations can be obtained from the images by virtue of the fact that the parts must fit together. If the user has geometric models of the parts, this information could also be used. The class definition is similar to that for the transport tool, with the expected modifications. We implemented this class on the PUMA robot, using adaptive visual servoing in 6 degrees of freedom as the visual-motor control mechanism. As in the case of the transport tool, the adaptive mechanism allows the task to be effectively carried out even with no prior information about the camera calibration or manipulator kinematics. Availability of a rough guess (provided by the system, not the user) improves performance substantially. Figure 4 shows an instantiation of this alignment class in use. Note, in contrast to the transport tool described previously, that we have been able to instruct the system to maintain orientation, so that, in this example, all the paths associated with tracked points are straight. Of course an alignment tool also permits us to specify rotations.

We have also implemented the alignment tool class on the Utah/MIT hand using the 6-DOF manipulation tool described earlier as a component or subtool. The implementation on the hand does not achieve the same accuracy as the implementation on the PUMA, primarily because the high degree of hysteresis and stiction in the hand requires that the gains in the visual servoing system be set low, and the match thresholds be set high in order to avoid oscillation. We are currently working to improve the performance by changing the control structure underlying the 6-DOF manipulator.

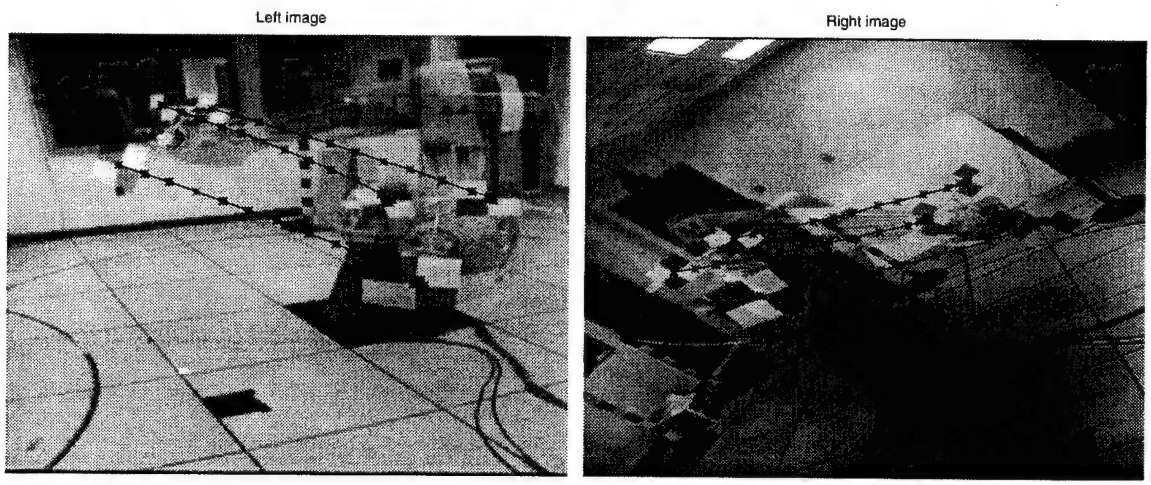


Figure 4: Use of an alignment tool for accurate positioning

4.5 A Visual Bending Tool

Our final example is an illustration of the complexity that can be handled using adaptive visual servoing techniques. Figure 5 shows the use of a bend tool, applied to a non-rigid foam beam. In this class, we require two 6-DOF manipulators with a total of 12 degrees of freedom. The user essentially supplies a couple of curves in binocular image space indicating the final shape of the beam, and possibly intermediate configurations in the case of extreme distortion. In the example shown, we tracked the location of points attached to the beam, and provided one intermediate configuration. Assuming that the desired shape is attainable, the tool effects it. Again, no model

information is used, and no prior constraints were placed on the use of the 12 degrees of freedom. This example would, in fact, be very difficult to model explicitly; the beam is non-rigid, and highly non-linear over the range of distortion exercised. Actually, this is a case where, depending on the sort of bends that are desired, it might be a good idea to implement a sub-tool to remap and reduce the raw freedoms of the two manipulators into a better conditioned space (e.g translation, rotation, bend and twist), as we did with the Utah-MIT hand. That the implementation worked without this is a tribute to the robustness of the adaptive visual servoing algorithm.

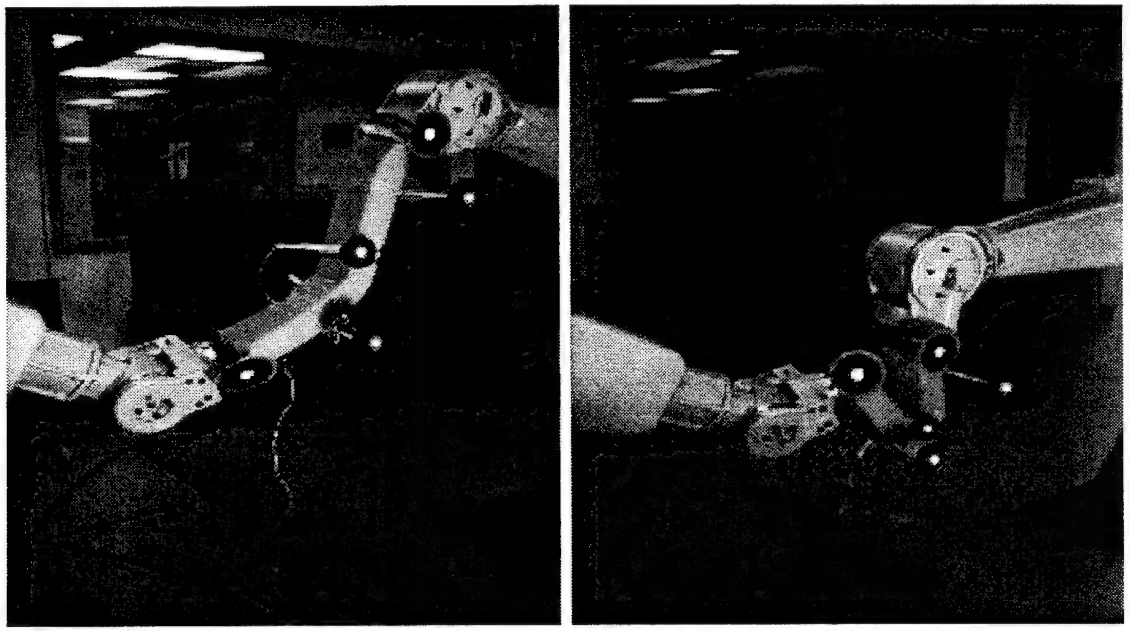


Figure 5: Application of a bending tool. Left: Initial configuration of two manipulators holding a foam beam Right: Achieved goal, specified via the tool instantiation parameters, which in this case represent binocular image positions of the white dots attached to the beam.

5 Integration into Applications

We have established a preliminary integration framework that allows tool classes to be instantiated and used sequentially in order to perform a more complex operation. If certain combined operations turn out to be frequently used, for example a pickup and insert operation, they can be defined as higher-level tool classes that incorporate lower level tools. We ultimately anticipate making heavy use of this aspect of the framework. In the current demonstrations however, we have programmed in terms of the individual tools described previously.

Figure 6 shows a shaped piece assembly operation performed under visual guidance by a PUMA Robot. The operation is performed in a tele-assistance environment, where the user specifies particular pieces, and how they should be inserted by pointing in an image. In this case, the system uses a pickup tool to acquire the indicated object, a transport tool to move it close to the user-designated insertion point, an alignment tool to line up the specified object corners with the specified hole corners, and finally, a local coordinate frame manipulator to perform the actual insertion. The system handles all initiation of visual tracking processes and instantiation and transformation of tools. The only input the user provides is to point to image locations.

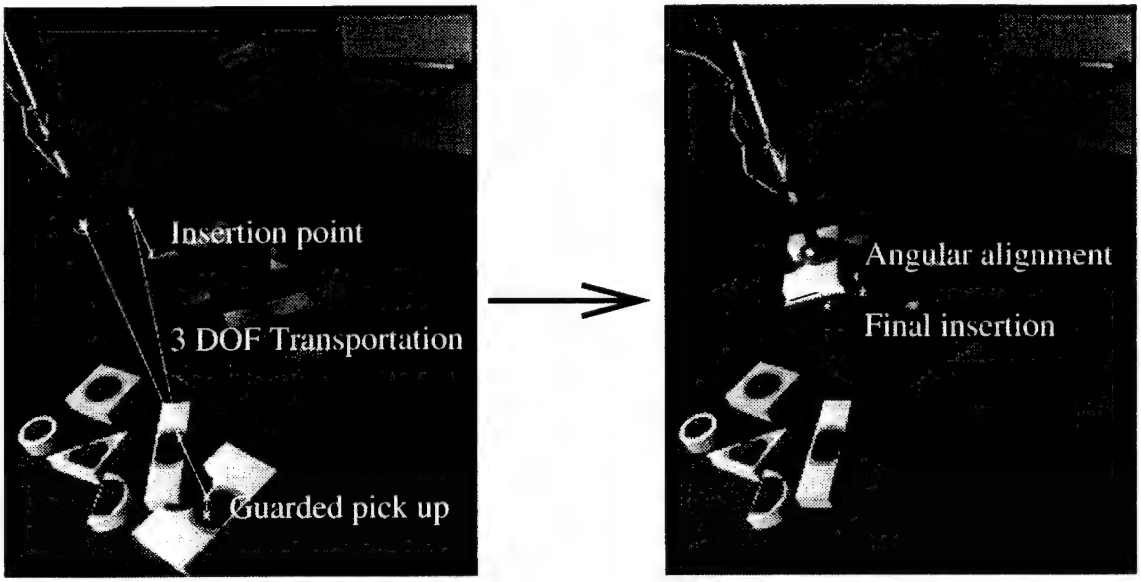


Figure 6: Pickup and insertion operation involving use of multiple tool classes. Right: pickup and transportation movements. Left: alignment and insertion.

As a second example, we performed the classic lightbulb insertion task. This operation made use of the eye-in-hand version of the visual transportation tool in order to pick up and insert the bulb into the socket. Repeated use of the grasp and manipulation tools on the Utah/MIT hand were used to screw the bulb into the socket until it lit up. For this operation, the manipulation tool was actually rolled into the definition of a more specialized rotation tool, which rotated the bulb about the appropriate axis (in this case, user supplied). Again, the system was used in tele-assistance mode, with the user input being supplied by pointing at the appropriate positions in live images. This operation is shown in Figures 7 and 8

6 Portability

One of the primary objectives of the work described here is to provide user-level primitives for using sensory-motor control that are portable across applications and across platforms. The examples we have presented make a fairly good case for portability across applications. Functions such as visually specified movement and alignment, and the need for robust grasping occur in large numbers of applications. What is not yet clear, is what the coverage is for a representative set of tasks in an application domain. We have performed fairly simple (for a human) assembly operations operations using only three or four tool classes. It is clear that more would be needed for assembling a space station or conducting an underwater salvage operation, even with a human operator in the loop. Clearly, a robot engaged in such a task would have access to a number of special hardware tools (wrenches, cutters etc.). Many of these could be used with basic visually guided movement and alignment operations, but others would need more specialized classes. We are currently investigating more complex assembly tasks in order to get an idea of how many classes would be required in a toolbox to cover some given fraction (e.g. 90%, 99%) of the necessary operations.

The case for general cross-platform portability is a little less clear, but here too we have some evidence. We have provided several examples of portability of virtual tool classes across platforms.

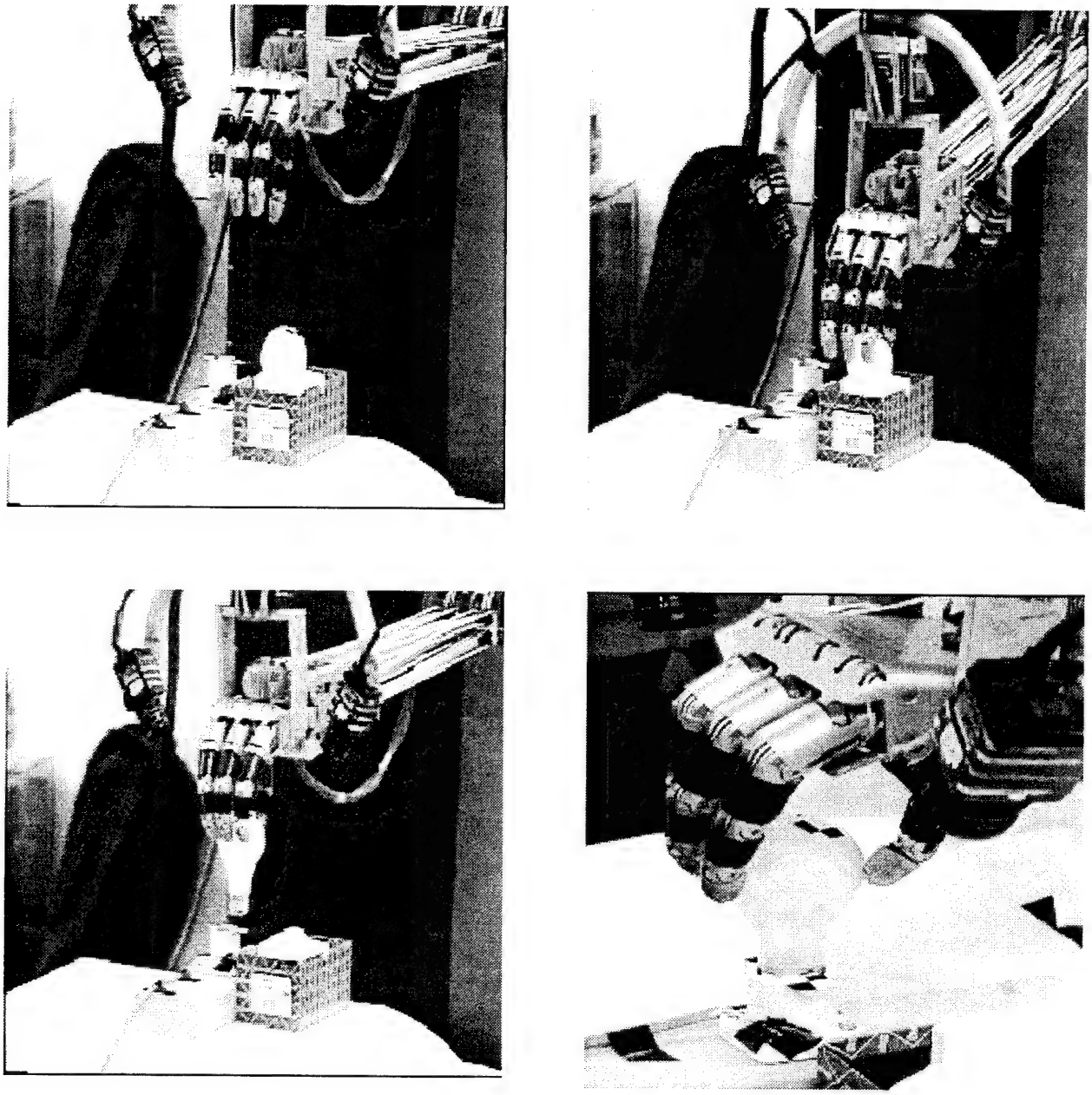


Figure 7: Side view of lightbulb insertion operation, showing binocular eye-in-hand camera configuration.

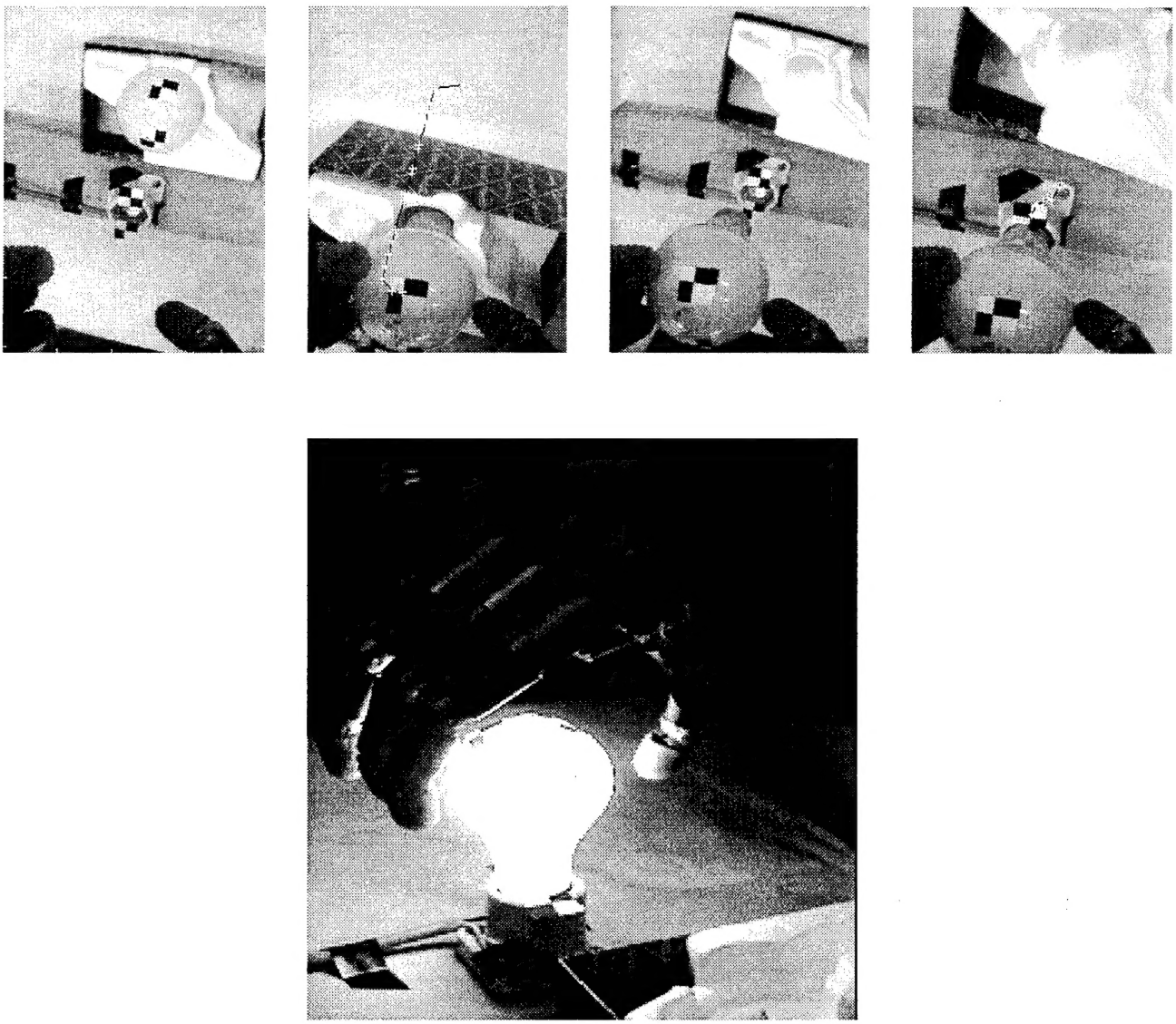


Figure 8: View of lightbulb insertion operation from eye-in-hand camera, showing pre-pickup, pickup, pre-insertion, pre-rotation, and completed phases.

The 6-DOF manipulation tool has been implemented on both the Utah/MIT hand and the PUMA, as has the visual alignment tool. These are radically different platforms, but the class semantics remained the same. The transportation tool was also ported, from a external observer, to an eye-in-hand configuration, again with preservation of the semantics (and in this particular case, no recoding of anything). Less dramatically, but still significant, we regularly swapped cameras, lenses, and observation positions without any need to readjust the software. With the exception of the accidental introduction of degenerate (or poorly conditioned) viewpoints, these sort of changes are completely transparent as far as the tools we are using are concerned.

7 Conclusions and Future Work

In this paper, we have described a construct termed a virtual tool class that provides a flexible interface to sensory-motor control, and can provide a substantial reduction in the effort needed to incorporate fairly sophisticated processes of this type into applications. Essentially the classes provide higher-level structuring elements that are more directly related to typical tasks that the raw device interfaces, and sufficiently general so as to be reusable across applications and platforms. We have described a number of specific tool classes, implemented them on a variety of platforms, and used them in coordinated fashion to perform higher level operations. In particular, we have described several visually-modulated tools that have robust implementations based on visual servoing.

Current research involves implementing additional tool classes needed for performing more complex assembly operations, and coming up with formal protocols of integrating sets of tool classes into usable toolboxes. With more complex assembly operations higher-level planning also becomes increasingly important, and we are investigating how higher-level organizational strategies can be incorporated into the framework. This involves not only high-level planning and description formalisms, but higher-level visual procedures such as recognition and object search as well.

References

[1] Yiannis Aloimomos. Active vision. *International Journal of Computer Vision*, 2:333–356, 1988.

[2] Yiannis Aloimonos. *Active Perception.* Lawrence Erlbaum, 1993.

[3] Michael Arbib, Thea Iberall, and Daniel Lyons. Coordinated control programs for movements of the hand. Technical Report TR, Dept of Computer Science, University of Massachusetts, 1985.

[4] Ruzena Bajcsy. An active observer. In *Proc. DARPA Image Understanding Workshop*, pages 137–147, San Diego, CA, January 1992.

[5] Ruzena Bajcsy and M. Campus. Active and exploratory perception. *CVGIP*, 56(1), July 1992.

[6] Dana H. Ballard. Reference frames for animate vision. In *Proc. IJCAI*, pages 1635–1641, August 1989.

[7] Dana H. Ballard and Christopher M. Brown. Principles of animate vision. *CVGIP*, 56(1):3–21, July 1992.

[8] Peter J. Burt. Smart sensing within a pyramid vision machine. *IEEE Procedings*, 76(8):1006–1015, 1988.

[9] P. Chongstitvatana and A. Conkie. Behavior-based assembly experiments using vision sensing. Technical Report 466, DAI, University of Edinburgh, 1990.

[10] K. Cleary. Incorporating multiple criteria in the operation of redundant manipulators. In *Proceedings of the 1990 IEEE International Conference on Robotics and Automation*, pages 618–624, Cincinnati, Ohio, 1990.

[11] A. Conkie and P. Chongstitvatana. An uncalibrated stereo visual servo system. Technical Report 475, DAI, University of Edinburgh, 1990.

[12] David J. Coombs, Tom J. Olson, and Christopher M. Brown. Gaze control and segmentation. In *Proc. AAAI Qualitative Vision Workshop*, Boston MA, August 1990.

[13] J. T. Feddema and G. C. S. Lee. Adaptive image feature prediction and control for visual tracking with a hand-eye coordinated camera. *IEEE Trans. Systems, Man and Cybernetics*, 20(5), 1990.

[14] J. T. Feddema and O. R. Mitchell. Vision guided servoing with feature-based trajectory generation. *IEEE Trans. on Robotics and Automation*, 5(5):691–670, 1989.

[15] Olac Fuentes and Randal C. Nelson. Morphing hands and virtual tools (or what good is an extra degree of freedom? Technical Report 551, University of Rochester, Computer Science Department, December 1994.

[16] Greg Hager, W. C. Chang, and S. Morse. Robot feedback control based on stereo vision: Towards calibration-free hand-eye coordination. Technical Report 992, CS Dept, Yale University, 1993.

[17] J. M. Hollerbach and K. C. Suh. Redundancy resolution of manipulators through torque optimization. In *Proceedings of the 1985 IEEE International Conference on Robotics and Automation*, pages 1016–1022, Nice, France, 1985.

[18] K. Hosoda and Minuru Asada. Versatile visual servoing without knowledge of the true jacobian. In *IROS*, August 1994.

[19] Thea Iberall. The nature of human prehesion: Three dextrous hands in one. In *Proc. ICRA*, pages 396–401, Raleigh NC, 1987.

[20] K. Ikeuchi and T. Suehiro. Towards an assembly plan from observation. In *Proc. Robotics and Automation*, 1992.

[21] Martin Jagersand and Randal C. Nelson. Adaptive differential feedback for uncalibrated hand-eye coordination and motor control. In *IROS*, Under Review, 1995.

[22] Kyros N. Kutulakos and Charles R. Dyer. Recovering shape by purposive viewpoint adjustment. In *Proc. CVPR*, pages 16–28, Champaign Il, June 1992.

[23] A. A. Maciejewski and C. A. Klein. Obstacle avoidance for kinematically redundant manipulators in dynamically varying environments. *The International Journal of Robotics Research*, 4(3):109–117, 1985.

[24] Scott McGhee, Tan F. Chan, Rajiv V. Dubey, and Reid L. Kress. Probability-based weighting of performance criteria for a redundant manipulator. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1887–1984, San Diego, California, 1994.

[25] Yoshihiko Nakamura, Hideo Hanafusa, and Tsuneo Yoshikawa. Task-priority based redundancy control of robot manipulators. *The International Journal of Robotics Research*, 6(2):3–15, 1987.

[26] Randal. C. Nelson. Qualitative detection of motion by a moving observer. *International Journal of Computer Vision*, 7(1):33–46, November 1991.

[27] Randal. C. Nelson. Vision as intelligent behavior: Research in machine vision at the university of rochester. *International Journal of Computer Vision*, 7(1):5–9, November 1991.

[28] Polly Pook and Dana Ballard. Teleassistance: Contextual guidance for autonomous manipulation. *Proc. AAAI*, August 1994.

[29] Marc H. Raibert. *Legged Robots That Balance.* The MIT Press, Cambridge, Massachusetts, 1986.

[30] Raymond D. Rimey and Christopher M. Brown. Where to look next using a bayes net: Incorporating geometric relations. In *Proc ECCV*, pages 542–550, May 1992.

[31] A. C. Sanderson and L. E. Weiss. Adaptive visual servo control of robots. In *Robot Vision, A. Pugh Editor*, 1983.

[32] L. E. Weiss. *Dynamic Visual Servo Control of Robots: An Adaptive Image-based Approach.* CMU, Phd Thesis, Pittsburgh PA, 1984.

[33] L. E. Weiss, A. C. Sanderson, and C. P Neumann. Dynamic sensor-based control of robots with visual feedback. *Journal of Robotics and Automation*, 3, October 1987.

[34] S. W. Wijesoma, D. F. H. Wolfe, and R. J. Richards. Eye-to-hand coordination for vision guided robot control applications. *Int. J. Robotics Research*, 12(1), 1993.

[35] David Wilkes and John Tsotsos. Active object recognition. In *Proc. CVPR*, pages 136–141, Champaign IL, June 1992.

[36] M. Inaba Y. Kuniyoshi and H. Inoue. Seeing, understanding, and doing human task. In *Proc. Robotics and Automation*, 1992.

[37] Tsuneo Yoshikawa. *Foundations of robotics : analysis and control.* The MIT Press, Cambridge, Massachusetts, 1990.

[38] B. H. Yoshimi and P. K. Allen. Active, uncalibrated visual servoing. In *ICRA*, Under Review, 1995.